QBone Scavenger Service Implementation for Linux

Mathieu Goutelle — Pascale Primet

Overview of the QBSS model

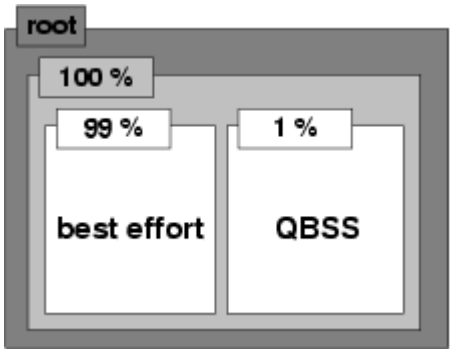
The QBSS model has been proposed by the **Internet2** QoS Working Group. It is part of the Non-Elevated Services promoted by the **Internet2** team. The concept is based on the following constat: most of the time, the capacity of a network is unused by the standard best effort service.

So, it can be a normal reaction to create a class which can use only the idle capacity of a network but stop using it as soon as the best effort class request more bandwidth. The complete specification of the QBSS model are available on the QBSS web site: `qbss-definition.txt` and `qbss-deployment-recommendation.txt`.

Our goal here is to implement this model in a standard Linux machine running as a router, using the DiffServ tools bundled with the kernel and the iproute package.

Presentation of our work

The implementation uses the CBQ scheduler included in the kernel. CBQ is particularly adapted for this kind of bandwidth sharing work. To follow the definition of QBSS and the deployment recommendation, we define two classes: one for best effort, the other for QBSS. 99% of the available output bandwidth is allocated to the first one, while we reserve the remaining 1% for the second class. But, the QBSS class is allowed to borrow the unused by best effort bandwidth.



The following script allows to configure the router to do the differenciation described above: bandwidth allocation, capacity of the QBSS class to borrow the unused bandwidth. It defines the parameters of the schedulers too. The signification of this parameters and the value of the important ones are explained and justified later.

```
#! /bin/bash
TC=/sbin/tc

# Initialisation
echo Initialisation...
$TC qdisc del root dev eth1 > /dev/null 2>&1

# qdisc configuration
echo qdisc configuration...
$TC qdisc add dev eth1 root handle 1: cbq bandwidth 100Mbit avpkt 1000b
$TC class add dev eth1 parent 1:0 classid  1:1 cbq bandwidth 100Mbit \
    rate 100Mbit maxburst 200 prio 1 allot 1514b avpkt 1000b bounded

# QBSS class definition
echo QBSS class definition...
$TC class add dev eth1 parent 1:1 classid 1:10 cbq bandwidth 100Mbit \
    rate  1Mbit maxburst  20 prio 2 weight 1Mbit allot 1514b avpkt
1000b

# best effort class definition
echo best effort class definition...
$TC class add dev eth1 parent 1:1 classid 1:20 cbq bandwidth 100Mbit \
    rate 99Mbit maxburst 200 prio 3 weight 1Mbit allot 1514b avpkt
1000b bounded

# Filters: need some customisation
echo Filters...
# cluster15 - QBSS
$TC filter add dev eth1 parent 1:0 prio 100 protocol ip \
    u32 match ip src 192.168.15.15 flowid 1:10
# cluster16 - best effort
$TC filter add dev eth1 parent 1:0 prio 100 protocol ip \
    u32 match ip src 192.168.16.16 flowid 1:20
# DS Field filter
#$TC filter add dev eth1 parent 1:0 prio 100 protocol ip \
#    u32 match ip tos 0x20 0xff classid 1:10
#$TC filter add dev eth1 parent 1:0 prio 100 protocol ip \
#    u32 match ip tos 0x00 0xff classid 1:20

echo Verification...
```

```
$TC -d class show dev eth1
```

We will first described the parameters whose value are the same for all the commands or which have no real influence on the scheduler (such as descriptors for example):

- **dev eth1** is the output interface of the router;
- **handle** allows us to associate a descriptor to the defined object, as a major index followed by a ":";
- **parent** is followed by the descriptor of the parent object, as two indexes (a major and a minor) separated with a ":";
- **avpkt** gives the average packet size (here 1ko for Ethernet frames).
- **classid** is followed by the defined class, as two indexes (a major and a minor) separated with a ":". The major index must be the same as the queuing discipline which ownes the class;
- **allot** defines the "chunkiness" of link sharing and is used for determining packet transmission tables. It differs slightly with the **allot** for class definition, where it specifies how many bytes the scheduler can dequeue during each round, weighted by the class weight. The value are the same nevertheless.

First, we erase all previous queuing discipline (If you don't want, comment this line). Then, the root queuing discipline is defined as **1:**, which will contains all the classes. Her **bandwidth** is set to the maximum available (100 Mbit/s). The "mother" class is defined after: the **bandwidth** parameter defines the available bandwidth of the parent object while the **rate** defines the bandwidth allocated to the current object.

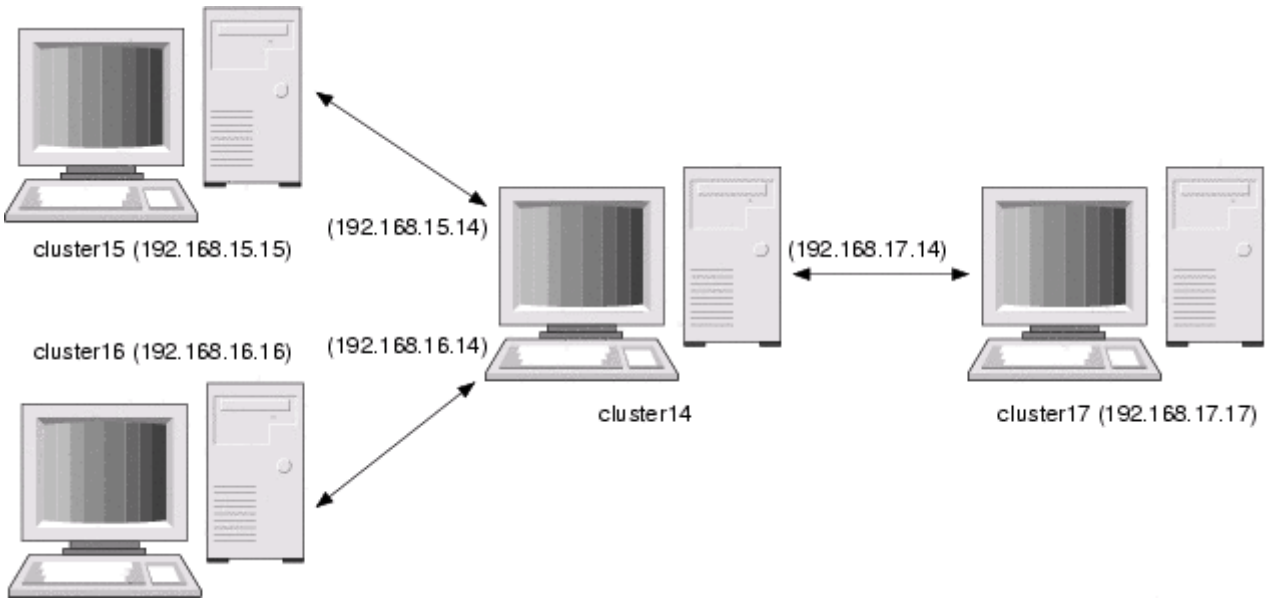
The two classes are then defined, as two leaf of the previous class. The QBSS class is allocated 1% of the capacity of the upper class, while best effort is allocated 99%. The **bounded** keyword mean that the best effort class cannot borrow bandwidth from the classes at the same level, in order to prevent QBSS from starvation. On the contrary, the router can allow QBSS to borrow the unused by best effort bandwidth, since the QBSS class isn't **bounded**.

The **weight** parameter gives the traffic proportion the scheduler will send from each class, like Weighted Round Robin. Since the two classes can access the same rate, I logically set both **weight** at the same value.

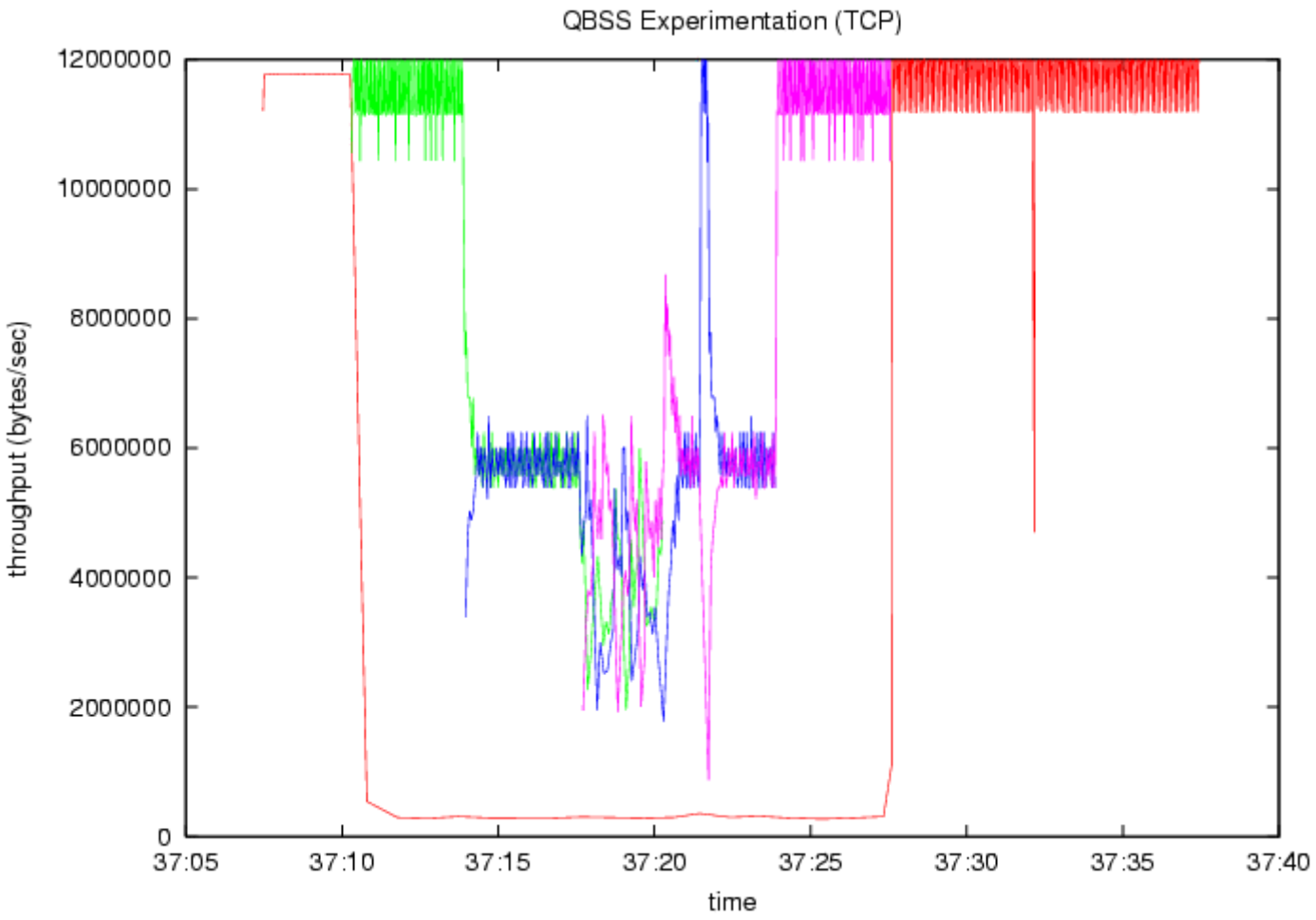
The two last lines set up the filters for separating the traffic in the correct class. Here, it is set up for the following validation. You can obviously do what you want here, especially treating the streams according to their DS field (`001000` for QBSS), like it is done in the comments at the end.

Validation

In order to validate our configuration, we have tried to do the same kind of experiments done by other teams with commercial routers. The testbed used is described on the figure below.



`cluster15` and `cluster16` will send TCP traffic to `cluster17` through the router (`cluster14`). According to the previous script, traffic coming from `cluster15` is treated as QBSS and traffic coming from `cluster16` is treated as best effort. `cluster15` will inject traffic during the whole experiment while `cluster16` will inject traffic later in the experiment, with three streams not synchronized in time. The results are recorded on the router using tcpdump and analysed using tcptrace.



As you can see, the QBSS stream (in red) has the expected behaviour: when it is alone in the router, the whole bandwidth is available for him. As soon as best effort traffic appears in the router, it gets only the 1% of reserved bandwidth and best effort can use the remaining 99%. Off course, when the three streams are together in the router, they share the bandwidth. As soon as all the three streams have disappeared, QBSS gets again the whole bandwidth.

Related work

As far as we know, experiments on QBSS use only commercial router (Cisco and Juniper). Links on the configurations to provide the QBSS differentiation on such routers and the associated validations are provided on the QBSS website.

References

- QBone Scavenger Service (QBSS)
- Linux Advanced Routing & Traffic Control
- References on CBQ, Sally Floyd


W3C
HTML
4.01
W3C
CSS